Knowl Inf Syst (2006) 10(3): 333–355
DOI 10.1007/s10115-006-0020-z

Knowledge and Information Systems

REGULAR PAPER

Ji Zhang · Hai Wang

Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance

Received: 3 December 2003 / Revised: 18 July 2005 / Accepted: 30 July 2005 /
Published online: 15 March 2006

Abstract In this paper, we identify a new task for studying the outlying degree (OD) of high-dimensional data, i.e. finding the subspaces (subsets of features) in which the given points are outliers, which are called their outlying subspaces. Since the state-of-the-art outlier detection techniques fail to handle this new problem, we propose a novel detection algorithm, called High-Dimension Outlying subspace Detection (HighDOD), to detect the outlying subspaces of high-dimensional data efficiently. The intuitive idea of HighDOD is that we measure the OD of the point using the sum of distances between this point and its k nearest neighbors. Two heuristic pruning strategies are proposed to realize fast pruning in the subspace search and an efficient dynamic subspace search method with a sample-based learning process has been implemented. Experimental results show that HighDOD is efficient and outperforms other searching alternatives such as the naive top–down, bottom–up and random search methods, and the existing outlier detection methods cannot fulfill this new task effectively.

Keywords Outlying subspace · High-dimensional data · Outlier detection · Dynamic subspace search

1 Introduction

Outlier detection is a classic problem in data mining that enjoys a wide range of applications such as the detection of credit card frauds, criminal activities, and exceptional patterns in databases. Outlier detection problem can be formulated as follows: Given a set of data points or objects, find a specific number of objects

J. Zhang (✉)
Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada
E-mail: jiz@cs.dal.ca

H. Wang
Sobey School of Business, Saint Mary's University, Halifax, Nova Scotia, Canada

that are considerably dissimilar, exceptional, and inconsistent with respect to the remaining data [9].

Numerous research works in outlier detection have been proposed to deal with the outlier detection problem defined earlier. They can broadly be divided into distance-based methods [13, 14, 19] and local density-based methods [7, 12, 17]. However, many of these outlier detection algorithms are unable to deal with high-dimensional data sets efficiently as many of them only consider outliers in the entire space. This implies that they will miss out the important information about the subspaces in which these outliers exist.

A recent trend in high-dimensional outlier detection is to use the evolutionary search method [3] where outliers are detected by searching for sparse subspaces. Points in these sparse subspaces are assumed to be the outliers. While knowing which data points are the outliers can be useful, in many applications, it is more important to identify the subspaces in which a given point is an outlier, which motivates the proposal of a new technique in this paper to handle this new task.

To better demonstrate the motivation of exploring outlying subspaces, let us consider the example in Fig. 1, in which 3 two-dimensional views of the high-dimensional data are presented. Note that point p exhibits different ODs in these three views. In the leftmost view, p is clearly an outlier. However, this is not so in the other two views. Finding the correct subspaces so that outliers can be detected is informative and useful in many practical applications. For example, in the case of designing a training program for an athlete, it is critical to identify the specific subspace(s) in which an athlete deviates from his or her teammates in the daily training performances. Knowing the specific weakness (subspace) allows a more targeted training program to be designed. In a medical system, it is useful for the Doctors to identify from voluminous medical data the subspaces in which a particular patient is found abnormal and therefore a corresponding medical treatment can be provided in a timely manner.

The major contribution of this paper is the proposal of *a dynamic subspace search algorithm, called High-Dimension Outlying subspace Detection (HighDOD), that utilizes a sample-based learning process to efficiently identify the subspaces in which a given point is an outlier*. Note that, instead of detecting outliers in specific subspaces, our method searches from the space lattice for the associated subspaces whereby the given data points exhibit abnormal deviations. To our best knowledge, this is the first such work in the literature so far. The main features of HighDOD include the following:

1. The outlying measure, OD, is based on the sum of distances between a data and its k nearest neighbors [4]. This measure is simple and independent of any underlying statistical and distribution characteristics of the data points;

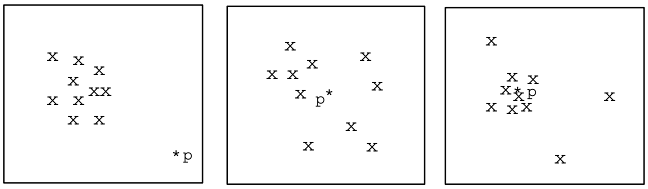

Fig. 1 Two-dimensional views of the high-dimensional data

2. Two heuristic pruning strategies are proposed to aid in the search for outlying subspaces;
3. A fast dynamic subspace search algorithm with a sample-based learning process is proposed;
4. The heuristic on the minimum sample size based on the hypothesis testing method is also presented.

The reminder of this paper is organized as follows. In Sect. 2, we survey the related work in the classic outlier detection problem. Section 3 discusses the basic notions and problem to be solved. In Sect. 4, we present our outlying subspace detection technique, called HighDOD, for high-dimensional data. Section 5 discusses the detailed algorithms of HighDOD. Experimental results are reported in Sect. 6. Section 7 concludes this paper.

2 Related work

Detecting outlying subspaces is a new task in the OD analysis for high-dimensional data, which has rarely been treated so far. However, we have witnessed intensive research efforts in dealing with a related traditional problem: detecting outliers in a given space or subspace. In this section, we will present a survey of literatures in this area to facilitate the analysis of applicability of these techniques in handling the new task to be introduced in this paper. Literatures on outlier detection can be classified into four major categories based on the techniques used, i.e. distribution-based methods, distance-based methods, density-based methods, and clustering-based methods.

Distribution-based methods [5, 10] rely on the statistical approaches that assume a distribution or probability model to fit the data set. Over 100 discordancy/outlier tests have been developed for different circumstances, depending on the parameter of data set (such as the assumed data distribution) and parameter of distribution (such as mean and variance), and the expected number of outliers [9, 13]. However, distribution-based methods suffer from some key drawbacks. First, they cannot be applied in a multi-dimensional scenario because they are univariate in nature. In addition, the lack of any prior knowledge regarding the underlying distribution of the data set makes the distribution-based methods difficult to use in practical applications. Finally, the quality of results cannot be guaranteed because they are largely dependent on the distribution chosen to fit the data.

The notion of distance-based outliers is also proposed [13, 14], i.e. DB(*pct, dmin*)-Outlier, which defines an object in a data set as a DB(*pct, dmin*)-Outlier if at least *pct%* of the objects in the data sets have the distance larger than *dmin* from this object. The notion of distance-based outlier is extended and the distance to the kth nearest neighbors of a point p, denoted as $Dk(p)$, is proposed to rank the point so that outliers can be more efficiently discovered and ranked [19]. The $Dk(p)$-outlier is further extended by considering for each point the sum of its k nearest neighbors [4]. Unlike distribution-based methods, distance-based methods do not rely on any assumed distribution to fit the data.

Recently, a density-based formulation scheme of outlier has been proposed [7]. This formulation ranks the OD of the points using *Local Outlier Factor (LOF)*. LOF of an object intuitively reflects the density contrast between its density and those of its neighborhood. Note that LOF ranks points by only considering the neighborhood density of the points, thus it may miss the potential outliers whose densities are close to those of their neighbors. The efficiency of this algorithm by proposing an efficient micro-cluster-based local outlier mining algorithm [12], but it still use LOF to mine outliers in data set.

The final category of outlier detection algorithm is clustering-based. So far, there are numerous studies on clustering, and a number of them are equipped with some mechanisms to detect outliers, such as CLARANS [16], DBSCAN [8], BIRCH [23], WaveCluster [22]. Clustering techniques tailored to subspace clustering for high-dimensional data includes DenClue [11], CLIQUE [1] and the technique proposed in [21]. Strictly speaking, clustering algorithms should not be considered as outlier detection methods because their objective is only to group the objects in data set such that clustering functions can be optimized. The aim to eliminate outliers in data set using clustering is only to dampen their adverse effect on the final clustering result.

The methods discussed in the aforementioned categories are not sufficiently applicable to high-dimensional data. Ref. [3] is the first work in high-dimensional outlier detection. It searches for sparse subspace using the evolutionary search method by first finding all the lower-dimensional projections that are locally sparse. The sparsity of a k-dimensional projection is measured by the so-called Sparse Coefficient of the corresponding k-dimensional cube. However, this Sparse Coefficient is not characterized by any upward or downward closure in the set of dimensions; hence, no pruning mechanism can be performed to find the sparse subspaces quickly.

Remark 1 All the existing outlier detection algorithms, regardless of in low- or high-dimensional scenario, invariably fall into the framework of detecting outliers in a specific data space, either in full space or subspace. We term these methods as "*space* $\rightarrow$ *outliers*" techniques. For instance, outliers are detected by first finding locally sparse subspaces [3], and the so-called Strongest/Weak Outliers are discovered by first finding the Strongest Outlying Spaces [14]. Our technique is novel in that it approaches the outlying analysis from a different prospective: finding the associated subspaces for each point in which this point is regarded as an outlier, which we can call it an "*outlier* $\rightarrow$ *spaces*" technique.

3 Outlying degree measure and problem formulation

Before we formally discuss our outlying subspace detection technique, we start with introduction of the OD measure that will be used in this paper and formulation of the new problem of outlying subspace detection we identify.

3.1 Outlying degree (OD)

For each point, we define the degree to which the point differs from the majority of the other points in the same space, termed the *outlying degree* (OD in short). OD is defined as the sum of the distances between a point and its k nearest neighbors in a data space [4]. Mathematically speaking, the OD of a point p in space s is computed as:

$$OD_s(p) = \sum_{i=1}^{k} Dist(p, p_i) | p_i \in KNNSet(p, s)$$

where $KNNSet(p, s)$ denotes the set composed by the k nearest neighbors of p in s. Note that the OD measure is applicable to both numeric and nominal data: for numeric data we use Euclidean distance, while for nominal data we use the simple match method. Mathematically, the Euclidean distance between two numeric points p_1 and p_2 is defined as $Dist(p_1, p_2) = [\sum((p_{1i} - p_{2i})/(Max_i - Min_i))^2]^{1/2}$, where Max_i and Min_i denote the maximum and minimum data value of the ith dimension. The simple match method measures the distance between two nominal points p_1 and p_2 as $Dist(p_1, p_2) = \sum |p_{1i} - p_{2i}|/t$, where $|p_{1i} - p_{2i}|$ is 0 if p_{1i} equals to p_{2i} and is 1 otherwise. t is the total number of attributes.

3.2 Problem formulation

We now formulate the new problem of outlying subspace detection for high-dimensional data as follows: given a data point or object, find the subspaces in which this data is considerably dissimilar, exceptional or inconsistent with respect to the remaining points or objects. These points under study are called *query points*, which are usually the data that users are interested in or concerned with.

A distance threshold T is utilized to decide whether or not a data point deviates significantly from its neighboring points. We call a subspace s is an outlying subspace of data point p if $OD_s(p) \geq T$. Note that the distance threshold T can either be a uniform threshold for all the subspaces or a different one for each of the subspaces. We call the former as the global distance threshold and the later as the local distance threshold.

3.3 Applicability of existing high-dimensional outlier detection techniques

The existing high-dimensional outlier detection techniques, i.e. find outliers in given subspaces, are theoretically applicable to solve the new problem identified in this paper. To do this, we have to detect outliers in all subspaces and a searching in all these subspaces is needed to find the set of outlying subspaces of p, which are those subspaces in which p is in their respective set of outliers. Obviously, the computational and space costs are both in an exponential order of d, where d is the number of dimensions of the data point. Such an exhaustive space searching is rather expensive in high-dimensional scenario. In addition, they usually only return the top-k outliers in a given subspace; thus, it is impossible to check whether or not p is an outlier in this subspace w.r.t the given distance threshold if p is not in this top-k list. This analysis provides an insight into the inherent difficulty of using the existing high-dimensional outlier detection techniques to solve the new outlying subspace detection problem.

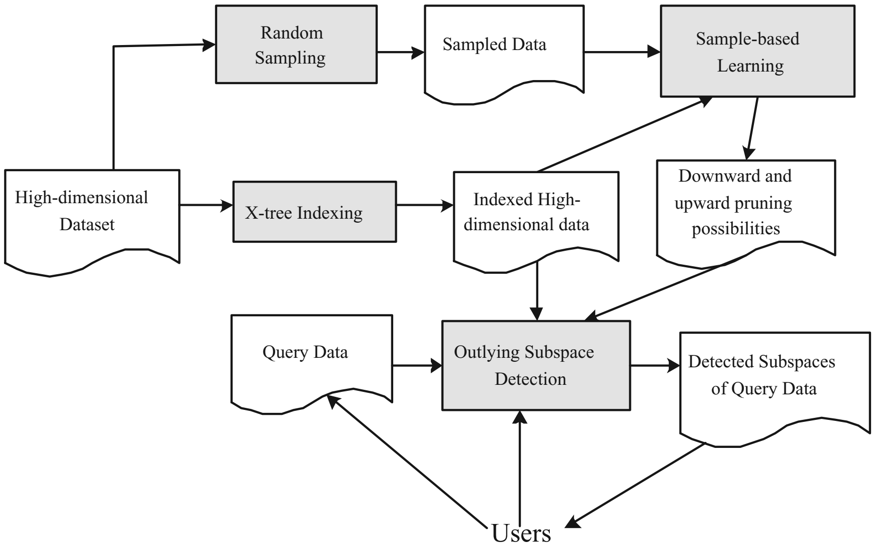
Random Sampling
Sampled Data
Sample-based Learning
High-dimensional Dataset
X-tree Indexing
Indexed High-dimensional data
Downward and upward pruning possibilities
Query Data
Outlying Subspace Detection
Detected Subspaces of Query Data
Users

Fig. 2 The overview of HighDoD

4 HighDOD

In this section, we present an overview of our HighDOD method (shown in Fig. 2). It mainly consists of four modules. The X-tree Indexing module performs X-tree [6] indexing of the high-dimensional data set to facilitate kNN search in every subspace. Sample-based Learning module randomly samples the data set and performs dynamic subspace search to estimate the downward and upward pruning probabilities of subspaces from 1 to d dimensions. Outlying Subspace Detection module uses the probabilities obtained in the Learning module to carry out a dynamic subspace search to find the outlying subspaces of the given query data point.

4.1 X-tree indexing

We utilize X-tree (eXtended node tree) [6] to index the high-dimensional data set in order to speedup the kNN queries in different subspaces (see Fig. 3). X-tree is an index structure that is designed to support efficient query processing of high-dimensional data. The introduction of X-tree is motivated by the problem with the R-tree-based index structures that the overlap of bounding boxes in the directory will increase as the dimension grows. The basic idea of X-tree is to use overlap-minimizing split and supernodes to keep the directory as hierarchical as possible and at the same time to avoid splits that will result in high overlap. It is a hybrid of linear array-like and R-tree like directory. The hierarchical organization is good for low dimensions, while in high dimensions, a linear organization is more efficient. For medium number of dimensionality, a dynamic organization of the tree such that the data producing high overlap are organized in a linear format, while those data can be organized hierarchically without causing too much overlap

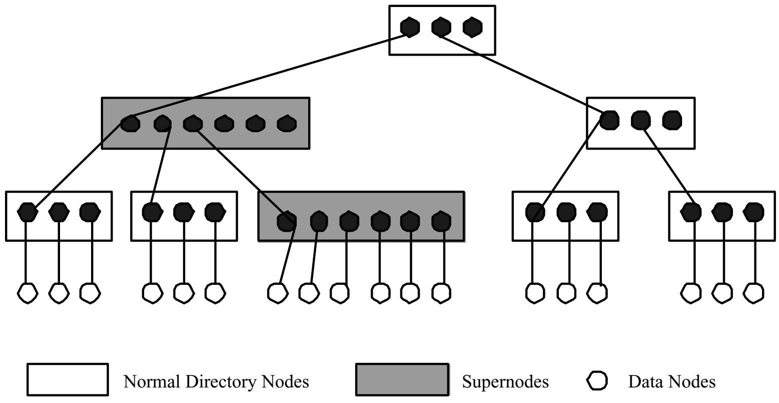

Fig. 3 X-tree

are organized in a hierarchy. X-tree has been shown experimentally outperform the well-known R*-tree and TV-tree for high-dimensional data by up to two orders of magnitude.

4.2 Subspace pruning

To find the outlying subspaces of a query point, we make use of the heuristics we devise to quickly detect the subspaces in which the point is not an outlier or the subspaces in which the point is an outlier. All these subspaces can be removed from further consideration in the later stage of the search process.

In our work, we utilize two classes of strategies for subspace pruning in the searching process, i.e. *Global-T pruning* and *Local-T pruning*. In the Global-T pruning, a global distance threshold T is used for delimiting outlying and non-outlying subspaces in the space lattice for a query data point. While in the Local-T pruning, different local distance thresholds are used for different subspaces required to evaluate in the searching process.

4.2.1 Global-T pruning strategy

OD maintains two interesting monotonic properties that allow the design of an efficient outlying subspace search algorithm in Global-T pruning. These two properties are based on the fact that the OD value of a point in a subspace cannot be less than that in its subset spaces. Mathematically, we have $OD_{s_1}(p) \geq OD_{s_2}(p)$ if $s_1 \supseteq s_2$. We first present the proof of this proposition as follows.

Proposition 1 *$OD_{s_1}(p) \geq OD_{s_2}(p)$ if $s_1 \supseteq s_2$.*

Proof Let a_k and b_k be the kth nearest neighbors of p in the an m-dimensional subspace s_1 and n-dimensional subspaces s_2, respectively ($1 \leq n \leq m \leq d$ and $s_1 \supseteq s_2$). $MaxDist_{s2}(p)$ is the maximum distance between p and a_i, $1 \leq i \leq k$, in the subspace s_2.

We have $Dist_{s1}(p, a_k) \geq Dist_{s1}(p, a_i)|_{1 \leq i \leq k}$. Since s_1 is a superset of s_2, we thus know $Dist_{s1}(p, a_i) \geq Dist_{s2}(p, a_i)|_{1 \leq i \leq k}$. This implies $Dist_{s1}(p, a_k) \geq Dist_{s2}(p, a_i)|_{1 \leq i \leq k}$, By definition of $MaxDist_{s2}$, we have $Dist_{s_1}(p, a_k) \geq MaxDist_{s2}(p) \geq Dist_{s2}(p, b_k)$. In other words, $Dist_{s1}(p, a_k) \geq Dist_{s2}(p, b_k)$. Likewise, it is hold that $Dist_{s1}(p, a_i) \geq Dist_{s2}(p, b_i)|_{1 \leq i \leq k}$, Since $OD_{s1}(p) = \sum_1^k Dist_{s1}(p, a_i)$ and $OD_{s2}(p) = \sum_1^k Dist_{s2}(p, b_i)$. We therefore conclude: $OD_{s1}(p) \geq OD_{s2}(p)$.

□

The corollary of the aforementioned proposition are the two monotonic properties given later.

Property 1 If a point p is not an outlier in a subspace s, then it cannot be an outlier in any subspace that is a subset of s.

Property 2 If a point p is an outlier in a subspace s, then it will be an outlier in any subspace that is a superset of s.

In the downward Global-T pruning strategy, we make use of Property 1 of OD to quickly prune away those subspaces in which the point cannot be an outlier. This is because if $OD_{s1}(p) < T$, then $OD_{s2}(p) < T$, where $s_1 \supseteq s_2$ and T is the distance threshold. In the upward Global-T pruning strategy, Property 2 of OD is utilized to detect those subspaces in which the point is definitely an outlier. The reason is that if $OD_{s2}(p) \geq T$, then $OD_{s1}(p) \geq T$.

The global distance threshold T is specified as follows:

$$T = C \sqrt{\sum_{i=1}^{d} \overline{OD}_{e_i}^2}, \quad \text{where } dim(e_i) = 1$$

where $\overline{OD}_{e_i}$ denotes the averaged OD value of points in the one-dimensional subspace e_i and C is a constant factor ($C > 1$). This specification stipulates that, in any subspace, only those points whose OD values are significantly larger than the average level in the full space are regarded as outliers. The average OD level in the full space is approximated by $\sqrt{\sum_{i=1}^{d} \overline{OD}_{e_i}^2}$ and the significance of deviation is specified by the constant factor C, normally we set $C = 2$ or 3.

4.2.2 Local-T pruning strategy

Let T_s be the local distance threshold for subspace s, and s_1 and s_2 be two subspaces that satisfy $s_1 \supset s_2$. In the downward Local-T pruning strategy, if $OD_{s1}(p) < T_{s_1}$, then s_2 will be pruned only when the following condition is satisfied:

$$\forall e_i, \quad OD_{e_i}(p) \leq \frac{1}{C} \cdot \overline{OD}_{e_i}, \quad \text{where } dim(e_i) = 1, \quad e_i \subseteq s_2$$

In the upward Local-T pruning strategy, if $OD_{s2}(p) \geq T_{s_2}$, then s_1 will be pruned only when the following condition is satisfied:

$$\forall e_i, \quad OD_{e_i}(p) \geq C \cdot \overline{OD}_{e_i}, \quad \text{where } dim(e_i) = 1, \quad e_i \subseteq s_1 - s_2$$

where C is the same constant factor used in the Global-T pruning. Being more conservative than the Global-T pruning, the two requirements mentioned earlier intuitively mean that the subspace s' which is a superset/subset of the current subspace s can be pruned for data point p only when the $OD(p)$ value in each of the one-dimensional subspaces that belong to s or the difference of the two subspaces (i.e. $s' - s$) is significantly smaller or larger than the corresponding average OD level.

Similar to the specification of the global distance threshold T, we can specify T_s, the distance threshold for a subspace s, as follows:

$$T_s = C\sqrt{\sum \overline{OD}_{e_i}^2}, \quad \text{where } dim(e_i) = 1, \quad e_i \subseteq s$$

The part of $\sqrt{\sum \overline{OD}_{e_i}^2}$ is the approximation of the average OD level in the subspace s.

Remark 2 By comparing the previous two pruning strategies, we can learn their pros and cons. Since the Global-T pruning strategy performs subspace pruning in a more aggressive way and may be able to prune a larger number of subspaces in each step than the Local-T pruning strategy, therefore it is faster to execute and more scalable to high-dimensional data set. However, the Global-T pruning strategy is limited in the situation that the average OD values of the points in different subspaces differ significantly, which makes it inaccurate to use a uniform distance threshold for all the subspaces. Adopting different distance thresholds for different subspaces is, therefore, more advantageous by taking into account the varied denseness/sparsity of points in different subspaces, which enables the Local-T pruning strategy to detect outlying subspaces with a higher level of accuracy. Still, it is comparatively slow and less scalable to high-dimensional data set. The selection between these two pruning strategies involves a tradeoff between speed/scalability and accuracy.

4.2.3 Saving factors of subspaces pruning

Now, we will compute the savings obtained by applying the pruning strategies during the search process quantitatively. Before that, let us first give three definitions.

Definition 1 Downward Saving Factor (DSF) of a Subspace.

The Downward Saving Factor of a m-dimensional subspace s is defined as the savings obtained by pruning all the subspaces that are subsets of s. In other words, the Downward Saving Factor of s, denoted as DSF(s), is computed as DSF(s) $= \sum_{i=1}^{m-1} C_m^i * i$, where C_m^i denotes the combinatorial number of choosing i items out of a total of m items.

Definition 2 Upward Saving Factor (USF) of a Subspace.

The Upward Saving Factor of an m-dimensional subspace s, denoted as USF(s), is defined as the savings obtained by pruning all the subspaces that are supersets of s. It is computed as USF(s) $= \sum_{i=1}^{d-m} [C_{d-m}^i * (m + i)]$.

Definition 3 Total Saving Factor (TSF) of a Subspace.

The Total Saving Factor of a m-dimensional subspace, in terms of a query point p, denoted as TSF(m, p), is defined as the combined savings obtained by applying the two pruning strategies during the search process. It is computed as follows:

$$\mathrm{TSF}(m,p) = \begin{cases} pr_{\mathrm{up}}(m,p) * f_{\mathrm{up}}(m) * \mathrm{USF}(m), & m = 1 \\ pr_{\mathrm{down}}(m,p) * f_{\mathrm{down}}(m) * \mathrm{DSF}(m) & \\ \quad + pr_{\mathrm{up}}(m,p) * f_{\mathrm{up}}(m) * \mathrm{USF}(m), & 1 < m < d \\ pr_{\mathrm{down}}(m,p) * f_{\mathrm{down}}(m) * \mathrm{DSF}(m), & m = d \end{cases}$$

where we have the following:

(1) $f_{\mathrm{down}}(m)$ and $f_{\mathrm{up}}(m)$ are the percentages of the remaining subspaces to be searched. Specifically, $f_{\mathrm{down}}(m) = C_{\mathrm{down\_left}}(m)/C_{\mathrm{down}}(m)$ and $f_{\mathrm{up}}(m) = C_{\mathrm{up\_left}}(m)/C_{\mathrm{up}}(m)$.
Let $dim(s)$ denote the number of dimensions for subspace s. $C_{\mathrm{down\_left}}(m)$ and $C_{\mathrm{up\_left}}(m)$ are computed as: $C_{\mathrm{down\_left}}(m) = \sum dim(s)$, where s is an unpruned or unevaluated subspace and $dim(s) < m$. $C_{\mathrm{up\_left}}(m) = \sum dim(s)$, where s is an unpruned or unevaluated subspace and $dim(s) > m$.
$C_{\mathrm{down}}(m)$ and $C_{\mathrm{up}}(m)$ are the total subspace search workload in the subspaces whose dimensions are lower and higher than m, respectively. Intuitively, $f_{\mathrm{down}}(m)$ and $f_{\mathrm{up}}(m)$ approximate the fraction of DSF and USF of an m-dimensional subspace that are potentially achievable in each step of the search process.

(2) $pr_{\mathrm{up}}(m, p)$ and $pr_{\mathrm{down}}(m, p)$ are the probabilities that upward and downward pruning can be performed in the m-dimensional subspace, respectively. In other words, for a m-dimensional subspace s, $pr_{\mathrm{up}}(m, p) = Pr(OD_s(p) \geq T)$ in the Global-T pruning and $pr_{\mathrm{up}}(m, p) = Pr(OD_s(p) \geq T_s)$ in the Local-T pruning. $pr_{\mathrm{down}}(m, p) = 1 - pr_{\mathrm{up}}(m, p)$ in both pruning strategies. A difficulty in computing the two prior probabilities, i.e. $pr_{\mathrm{up}}(m, p)$ and $pr_{\mathrm{down}}(m, p)$, is that their values are unknown if there lacks any prior knowledge of the data set. To overcome this difficulty, we first perform a sample-based learning process to obtain some knowledge about the data set and then apply this knowledge in the later subspace search for each query point. The estimates of $pr_{\mathrm{up}}(m, p)$ and $pr_{\mathrm{down}}(m, p)$ can be viewed as an approximated estimation of the linkage in the data probability distribution of the data set involved.

4.3 Sampling-based learning

We adopt a sample-based learning process to obtain some knowledge about the data set before subspace search of the query points are performed. This is desirable when the data set is large so that learning the whole data set becomes prohibitive. The task of performing this sampling-based learning is twofold: first, we will have to estimate $\overline{OD_{s_i}}$ which will be used in specifying the global and local distance thresholds and the requirements in the Local-T pruning. Secondly, we will have to compute the two priors $pr_{\text{up}}(m, p)$ and $pr_{\text{down}}(m, p)$. In this learning process, a small number of points are randomly sampled from the data set.

At first, the subspace searches are performed in the d one-dimensional subspaces s_i on all the sampling data and $\overline{OD_{s_i}}$ is computed as the average OD values of all sampling points in subspace s_i, i.e.

$$\overline{OD_{s_i}} = \frac{1}{S}\sum_{j=1}^{S} OD_{s_i}(sp_j)$$

where S is the number of sampling points and sp_j denotes the ith sampling point.

Secondly, the subspace searches are performed in the lattice of data space on the sampling data. For each sampling point *sp*, we have the following initial specifications regarding the two priors $pr_{\text{up}}(m, p)$ and $pr_{\text{down}}(m, p)$:

$$\begin{array}{ll} pr_{\text{up}}(m, sp) = pr_{\text{down}}(m, sp) = 0.5, & 1 < m < d \\ pr_{\text{up}}(m, sp) = 1 \quad \text{and} \quad pr_{\text{down}}(m, sp) = 0, & m = 1 \\ pr_{\text{up}}(m, sp) = 0 \quad \text{and} \quad pr_{\text{down}}(m, sp) = 1, & m = d \end{array}$$

This initialization implies that we assume equal probabilities for upward and downward pruning in the subspaces of any dimension, except 1 and d, for each sampling point at the beginning. After all the m-dimensional subspaces have been evaluated for *sp*, the $pr_{\text{up}}(m, sp)$ and $pr_{\text{down}}(m, sp)$ are computed as the percentages of m-dimensional subspaces s in which $OD_s(sp) \geq T$ and $OD_s(sp) < T$, respectively. The average pr_{up} and pr_{down} values of subspaces from 1 to d dimensions can be obtained as follows:

$$\overline{pr_{\text{up}}(m)} = \frac{1}{S}\sum_{i=1}^{S} pr_{\text{up}}(m, sp_i)$$

$$\overline{pr_{\text{down}}(m)} = \frac{1}{S}\sum_{i=1}^{S} pr_{\text{down}}(m, sp_i)$$

where we have $\overline{pr_{\text{down}}(1)} = \overline{pr_{\text{up}}(d)} = 0$.

For each query point p, we set $pr_{\text{up}}(m, p) = \overline{pr_{\text{up}}(m)}$ and $pr_{\text{down}}(m, p) = \overline{pr_{\text{down}}(m)}$ in the computation of TSF(m, p) of the query point p.

Remark 3 There might be a misunderstanding that the sampling technique will fail here because the outliers are rare in the data set. Recall that we are trying to detect outlying subspaces of query points, not outliers. Every point can become query point and every query point will have its outlying subspaces, if its set of outlying subspaces is not empty. Hence, the outlying subspaces can be regarded as a global property for all the points and a sample of sufficient size will make sense in the learning process.

4.4 Dynamic subspace search

In HighDOD, we use a dynamic subspace search method to find the subspaces in which the sampling points and the query points are outliers. The basic idea of

Table 1 Computation of Dyna-CSF

Dimension of subspaces	TSF
(a)	
4	**28**
3	13
2	12
1	19
(b)	
3	9
2	10
1	**17.7**
(c)	
3	**2.7**
2	2

the dynamic subspace search method is to commence search on those subspaces with the same dimension that has the highest TSF value. As the search proceeds, the TSF of subspaces with different dimensions will be updated and the set of subspaces with the highest TSF values are selected for exploration in each subsequent step. The search process terminates when all the subspaces have been evaluated or pruned. Note that the only difference between the dynamic subspace search method used on the sample points and query points lies in the decision of values of $pr_{\text{up}}(m, p)$ and $pr_{\text{down}}(m, p)$: *For sample points, we assume an equal probability of upward and downward pruning (referring to Section 4.3), while for query points we use the averaged probabilities obtained in the learning process.*

Example 1 Now, we give an example to illustrate how the TSF of a subspace is computed at each step of the search process using Global-T pruning strategy. Without loss of generality, we suppose, for a sample point *sp*, the $OD(sp)$ values of the subspaces in the boxes are larger than T, while the $OD(sp)$ values of the subspaces underlined are smaller than T, as shown in Fig. 4. We compute the TSF for subspaces with different dimensions (1–4) in each step of the search process (as shown in Table 1(a)–(c)). In each step, we select the dimension that has the maximum TSF value, which are highlighted in each of the tables. The order of subspaces in the search process is $4 \rightarrow 1 \rightarrow 3 \rightarrow 2$, meaning that the four-dimensional subspaces are searched first, followed by one- and three-dimensional subspaces. The two-dimensional subspaces are searched at last.

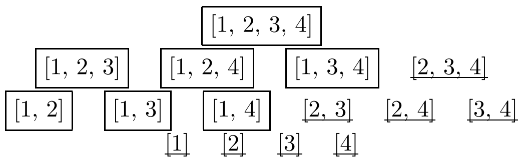

Fig. 4 Subspaces of a four-dimensional full space

4.5 Minimum sampling size for training data set

Recall that the sampling method is utilized to obtain a training data set that can be used to pre-compute the prior probabilities of upward and downward pruning, namely $\overline{pr_{\text{up}}(m)}$ and $\overline{pr_{\text{down}}(m)}$ $(1 \leq m \leq d)$. As such, samples of different sizes will only affect the pruning efficiency of the algorithm. They will not change the number of subspaces found.

With this in mind, we now wish to determine the minimum sample size to accurately predict $\overline{pr_{\text{up}}(m)}$ and $\overline{pr_{\text{down}}(m)}$ with certain degree of confidence. We denote X as the sample point that can be expressed as an S-dimensional vector as $X = [x_1, x_2, \ldots, x_S]$ where S is the size of the sample. Each data in the sample is a d-dimensional vector as $x_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,d}]^T$ where $x_{i,j}$ denote the value of jth dimension of ith data in the sample. Applying dynamic subspace searching on sampling points, for each dimension m, we obtain

$$Y_{\text{down}}(m) = [pr_{\text{down}}(m, sp_1), pr_{\text{down}}(m, sp_2), \ldots, pr_{\text{down}}(m, sp_S)]\ (1 \leq m \leq d)$$

We use the S measurements, $pr_{\text{down}}(m, sp_i)(1 \leq i \leq S)$ as the training data to estimate the mean of $pr_{\text{down}}(m)$. We estimate the sample size by constructing the confidence interval of the mean of $pr_{\text{down}}(m)$. Specifically, to obtain a $(1-\alpha)$-confidence interval, the minimum size of a random sample is given as follows [15]:

$$S_{\min}(m) = \left[\frac{t_{\alpha/2} * \sigma'_m}{\delta*}\right]^2$$

where σ'_m denotes the estimated standard deviation of pr_{down} in the mth dimension using the training points that is defined as:

$$\sigma'_m = \sqrt{\sum_{i=1}^{S} (pr_{\text{down}}(m, sp_i) - \overline{pr_{\text{down}}(m, sp)})^2/(S-1)}$$

$\delta*$ denotes the half-width of the confidence interval.

Note that the value of σ'_m varies for different m. Let $\sigma'_{\max} = max(\sigma'_m)(1 \leq m \leq d)$, the minimum sample size $S_{\min}$ that satisfies respective minimum sample size requirement of each dimension is computed as:

$$S_{\min} = \left[\frac{t_{\alpha/2} * \sigma'_{\max}}{\delta*}\right]^2$$

Similar reasoning applies to $\overline{pr_{\text{up}}(m)}$, since $\overline{pr_{\text{up}}(m)} = 1 - \overline{pr_{\text{down}}(m)}$.

5 Algorithms

The algorithm of dynamic subspace search is presented in Fig. 5. In this algorithm, *SetDim* stores the number of dimension of the subspaces (from 1 to d). A non-empty *SetDim* indicates that there are still some subspaces to be searched. The new TSF of the remaining dimensions are computed (calling function *ComputeDynaTSF()*) and an additional pass of the subspace search is performed. The

```
Algorithm HighDoD
Input: $p$, Dataset $D$, $T$ and $k$
Output: All the subspaces in which $p$ is an outlier
$SetDim \leftarrow \emptyset$;    $SetDetectSS \leftarrow \emptyset$;
FOR $i \leftarrow 1\, to\, d\, DO$   SetDim$\cup = i$;
WHILE ($SetDim \neq \emptyset$) THEN {
  ComputeDynaTSF($SetDim$);
  $CurrDim \leftarrow$ MaxiDynaTSF($SetDim$);
  $SetDetectSS \cup \leftarrow$ SubspaceSearch($CurrDim$);
  $SetDim$-=$CurrDim$;}
Return ($SetDetectSS$);
```

Fig. 5 Algorithm of dynamic subspace search

dimension of the subspace with the maximum TSF in the current step is stored in the variable *CurrDim* (calling function *MaxiDynaTSF()*). All the subspaces detected (calling function *SubspaceSearch()*) are saved in *SetDetectSS*. The current dimension of subspaces is deleted from *SetDim* and the whole program terminates when *SetDim* becomes empty.

Note that the algorithm presented in Fig. 5 can run on both the sampling and the query points. The only difference lies in the function *ComputeDynaTSF()*. Here, the sampling points use the pre-defined probabilities as the priors, while the query points use the probabilities as the priors obtained in the sample-based learning process. The function of *ComputeDynaTSF()* dynamically computes the TSF values in the subspace searching process.

The function SubspaceSearch() (see Fig. 6) returns all the detected subspaces of the *CurrDim* dimension. It displays a generic skeleton of the function Subspace Search() for subspace searching using either the Global-T or the Local-T pruning

```
Algorithm SubspaceSearch
Input: Dimension of subspaces to be searched $CurrDim$.
Output: All the $CurrDim$ dimensional subspaces in which $p$ is an outlier.
$SetDetectSS\_CurrDim \leftarrow \emptyset$;
IF ($CurrDim = 1$) THEN
  FOR each unpruned subspace $ss$ of $CurrDim$ dimension DO
    IF($OD^k_{ss}(p) \geq T$) THEN {
      $SetDetectSS\_CurrDim \cup = ss$;
      PruneExistSS(Superset($ss$)); }
ELSE IF ($CurrDim = d$) THEN
    IF ($dim(ss) = d$ and $OD^k_{ss}(p) \geq T$) THEN
      $SetDetectSS\_CurrDim \cup = ss$;
    ELSE PruneExistSS(Subset($ss$));
ELSE
  FOR each unpruned subspace $ss$ of $CurrDim$ dimension DO
    IF ($OD^k_{ss}(p) \geq T$) THEN {
      $SetDetectSS\_CurrDim \cup = ss$;
      PruneExistSS(Superset($ss$)); }
    ELSE PruneExistSS(Subset($ss$));
Return ($SetDetectSS\_CurrDim$);
```

Fig. 6 The generic algorithm of searching subspaces of a given dimension

strategy. Note that there are two differences between the exact implementation of the function for the two pruning strategies: (1) T represents the global distance threshold in the Global-T pruning strategy and the local distance threshold for each subspace in the Local-T pruning strategy; (2) In the Local-T pruning strategy, the algorithm has to check the satisfiability of the pruning requirements within the functions $PruneExistSS(superset(ss))$ and $PruneExistSS(subset(ss))$ before the pruning is performed, while the Global-T pruning strategy does not have to.

6 Experimental results

In this section, we will carry out extensive experiments to test the efficiency of outlying subspace detection and the effectiveness of outlying subspace compression in HighDOD. Synthetic data sets are generated using a high-dimensional data set generator and four real-life high-dimensional data sets from the UCI machine learning repository, which have been used in [3] for performance evaluation of their high-dimensional outlier detection technique, are also used.

In the synthetic data generator, we can specify the number of instances (tuples) (N) and dimensions (d) of the data set generated. We will further specify the number of the intervals (N_{I}) and the maximum (max) and minimum (min) values for each dimension of the data set. The length for each interval will be $(max - min)/(N_{\mathrm{I}})$. For each dimension, N/N_{I} distinct instances are generated based on the Gaussian distribution in each of the N_{I} intervals. (The projection of high-dimensional data within small intervals in each dimension can be reasonably assumed to fit Gaussian distribution [3].) In this way, we will be able to obtain N distinct instances for each dimension. The data set generated generally displays dense and sparse regions in the data space, which serves as an ideal test-bed for our experiments. Specifically, we specify $N_{\mathrm{I}} = 10, min = 0, max = 100$ for each dimension in our experimental setting.

All the algorithms are implemented on a 1.8 GHz Pentium PC with 256 MB of main memory running on Windows 2000. In all the experiments, we set k, the number of neighbors we are interested in, to 10.

6.1 Efficiency study

Since the existing high-dimensional outlier detection techniques fail to handle the new outlying subspace detection problem, we thus choose to compare the efficiency of several subspace search methods, i.e. top–down, bottom–up, random and dynamic subspace search, instead.

These searching methods aim to find the outlying subspaces of the given query data using various searching strategies. The top–down search method only employs a downward pruning strategy, while the bottom–up search method only uses an upward pruning strategy. The random search method, the "headless chicken" search alternative, randomly selects the layer in the lattice for search without replacement in each step. The dynamic search method, a hybrid of upward and downward search, computes the TSF of all subspaces of different dimensions and selects the best layer of subspaces for search. To evaluate the efficiency of the

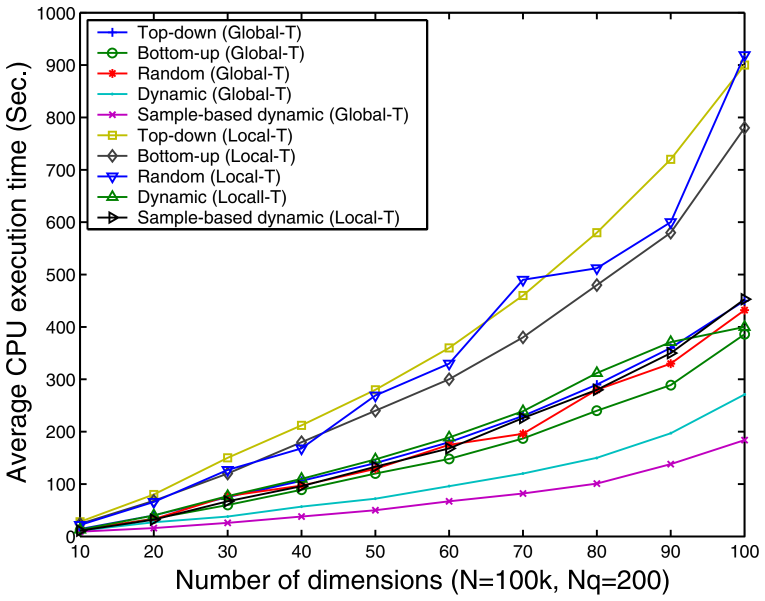

Fig. 7 Execution time when varying dimension of data

sample-based learning process, we run the dynamic search algorithm with and without incorporating the sample-based learning process. We implement the previous five searching methods with each of the two pruning strategies, which results in a total of 10 searching methods. Note that the execution times shown in this section are the average time spent in processing each point in the learning and query process.

6.1.1 Effect of dimensionality

First, we investigate the effect of dimensions on the average execution time of HighDOD (see Fig. 7). We can see that the execution time of all the 10 methods increase at an exponential rate since the number of subspaces increases exponentially as the number of dimension goes up, regardless of which searching and pruning strategy is utilized. On a closer examination, we see that (1) the execution time of top–down and bottom–up search methods increase much faster than the dynamic search method; (2) when using the sample-based learning process, the dynamic search method performs better than the method without using the sample-based learning process; (3) the execution times of the methods using the Global-T pruning strategy increase much less slowly compared with the those methods using the Local-T pruning strategy. This is because the Local-T pruning strategy is more conservative than the Global-T pruning strategy and therefore fewer subspaces can be pruned in each step of the searching process.

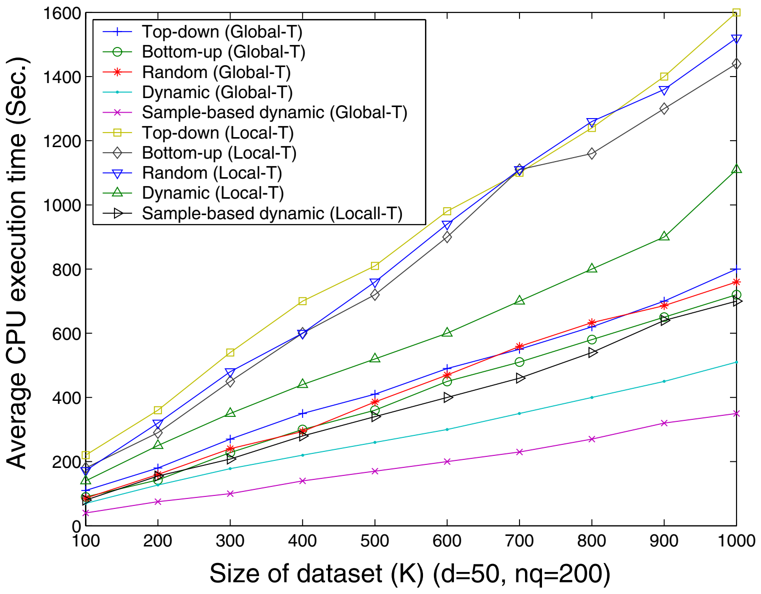

Fig. 8 Execution time when varying size of data set

6.1.2 *Effect of data set size*

Second, we fix the number of dimensions at 50 and vary the size of data sets from $100k$ to $1000k$. Figure 8 shows that the average execution times using the 10 methods to process each query point are approximately linear with respect to the size of the data set. Similar to results of the first experiment, the dynamic search method with sample-based learning process and Global-T pruning strategy gives the best performance.

6.1.3 *Effect of number of query points*

Next, we vary the number of query points N_q. Figure 9 shows the results of the five searching method using the Local-T pruning strategy only. It is interesting to note that when N_q is large, dynamic search method with sample-based learning process gives the best performance. However, when N_q is small, it is better to use dynamic search without sample-based learning. The reason is because when the number of query points is small, the saving in computation by using the learning process is not sufficient to justify the cost of the learning process itself.

6.1.4 *Effect of sample size*

We also investigate the effect of the number of sampling points, S, used in the learning process. A large S gives a more accurate estimation of the possibilities of upward and downward pruning in subspaces, which in turn, helps to speedup the search process. However, a large S also implies an increase in the computation during the learning process, which may increase the average time spent in the

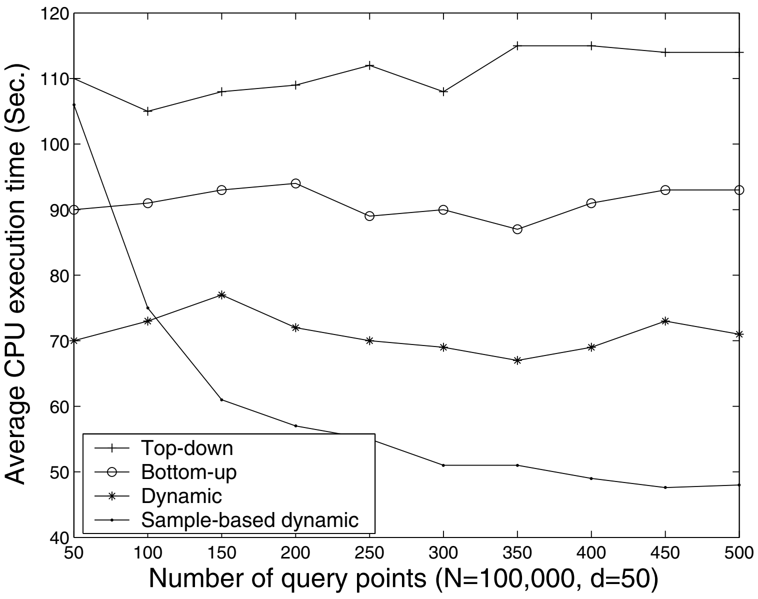

Fig. 9 Execution time when varying the number of query points

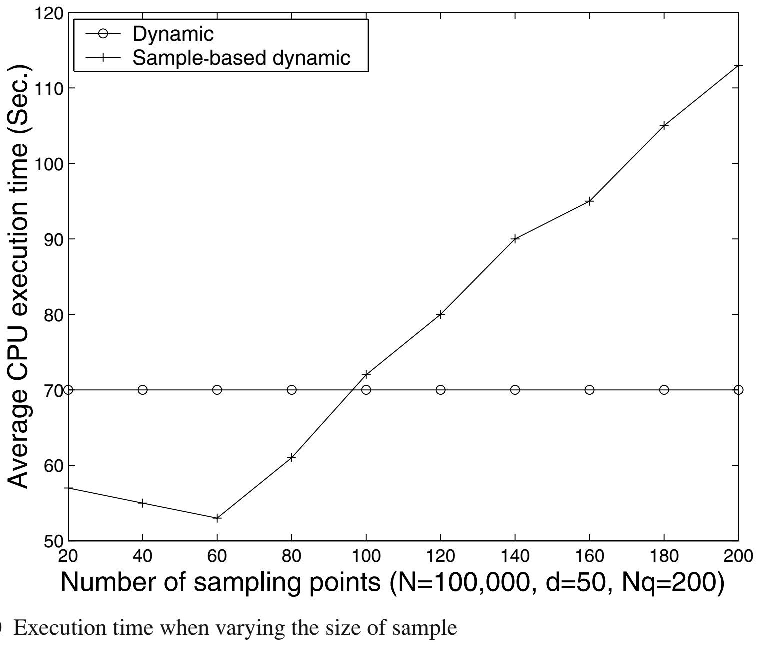

Fig. 10 Execution time when varying the size of sample

whole detection process. As shown in Fig. 10, the execution time is first decreased when the number of sampling points is small, this is because the prediction of possibility is not accurate enough, which cannot greatly speedup the later searching process. When the sample size increases, the prediction of the possibilities are sufficiently accurate, therefore any larger size of sample will no longer contribute to the speedup of the search process, but only increase the execution time as a whole. The horizontal dot-line in Fig. 10 indicates the execution time when dynamic subspace search without sample-based learning is employed.

6.1.5 Results on real-life data sets

Finally like [3], we evaluate the practical relevance of HighDOD by running experiments on five real-life high-dimensional data sets in the UCL machine learning repository. The data sets range from 8 to 160 dimensions. Table 2 shows the results of the five search methods using the Local-T pruning strategy. It is obvious that dynamic search with sampling-based learning process works best in all the real-life data sets. Furthermore, using dynamic subspace search alone is faster than top–down bottom–up or random search methods by approximately 20%, while incorporating sample-based learning process into dynamic subspace search further reduces the execution time by about 30%.

6.2 Effectiveness study

In this part, we will show that the existing outlier detection techniques (we term them “$space \rightarrow outliers$” techniques) will break down when trying to perform the task of finding the subspaces in which given points are outliers. We will first present two definitions that will be used in the experiment evaluation in this part.

Definition 4 Outlying Strength of points.

The Outlying Strength of a point p, denoted as $OS(p)$, is defined as the percentage of number of subspace in which p is a outlier against the total number of subspaces, i.e.

$$OS(p) = \frac{\text{No. of subspaces in which } OD_s(p) \geq T}{\text{No. of subspaces}}$$

Definition 5 Detecting ability of “$space \rightarrow outliers$” techniques.

Table 2 Results of running five methods on real-life data sets (average CPU time in seconds for each query point)

Data sets (dimensions)	Top–down	Bottom–up	Random	Dynamic	Sample + dynamic
Machine (8)	56	49	58	41	32
Breast cancer (14)	165	176	150	121	110
Segmentation (19)	251	237	256	222	197
Ionosphere (34)	472	477	456	414	387
Musk (160)	5203	4860	5002	4389	3904

We measure the ability of "$space \rightarrow outliers$" techniques in finding the subspaces in which given points are outliers, denoted as φ, by computing the percentage of the number of given points that appear in the top n subspaces obtained by the method against the total number of given points, i.e.

$$\varphi = \frac{\text{No. of given points that appear in the subspaces selected}}{\text{No. of given points}}$$

The setup of this experiment is specified as follows. We first select a fixed number of points (we select 50 in our experiment) for each different group of Outlying Strength ranging from 10 to 80%. (The information of Outlying Strength of points can be obtained using HighDOD in advance.) We then apply the evolutionary-based search method, the latest member of the "$space \rightarrow outliers$ techniques, on all these selected points and compute the value of φ based on points in each group. We adopt two ways for selecting the subspaces for study, i.e. randomly selecting n subspaces from the whole space lattice and selecting the top-k subspaces returned by the evolutionary-based search method. We vary the value of n, the number of subspace selected, from 20 to 100 in this experiment. The objective of this experiment is to test the effectiveness of the evolutionary-based search method in finding the subspaces of query points under different Outlying Strengths of given points and different number of subspaces returned by this method.

The result is shown in Tables 3 and 4. We can see, from the two tables, that the evolutionary-based search method performs poorly in finding the subspaces in which points are outliers. For overwhelming majority of the points, this method cannot return any subspaces in which these points are true outliers. Even when a point appears in the top n subspaces obtained by this method, the method can only find the subspaces within the n subspaces returned rather than the whole spectrum of subspaces. Increasing the number of subspaces returned can theoretically improve the performance, but such increasing in the number of subspaces will invariably increase the computation exponentially that render it almost infeasible in practice. The reason that the evolutionary-based search method, also including other "$space \rightarrow outliers$" techniques, performs poorly in this experiment is because the subspaces that are spare are not necessarily, in most cases, the right subspaces in which the given points are outliers.

Table 3 Values of φ (%) of the evolutionary-based search method with randomly selected subspaces

No. of subspaces	10–20%	20–30%	30–40%	40–50%	50–60%	60–70%	70–80%
20	0	0	0	0	0	2	4
60	0	0	0	0	0	4	6
100	0	0	0	0	3	7	11

Table 4 Values of φ (%) of the evolutionary-based search method with top-k subspaces

No. of subspaces	10–20%	20–30%	30–40%	40–50%	50–60%	60–70%	70–80%
20	0	0	0	0	0	3	5
60	0	0	0	0	1	4	6
100	0	0	0	0	2	9	12

7 Conclusions

In this paper, we propose a novel algorithm, called HighDOD, to address the new problem of detecting outlying subspaces for high-dimensional data. In HighDOD, two heuristics for fast pruning in the subspace search and a dynamic subspace search method with a sample-based learning process are used. Experimental results justify the efficiency and effectiveness of outlying subspace searching in HighDOD, in comparison with other search alternatives and the existing outlier detection techniques. We believe that HighDOD is useful in revealing new and interesting knowledge in outlying analysis of high-dimensional data and can be potentially used in many practical applications.

By no means dismissing their advantages, we aim to show the "$space \rightarrow outliers$" techniques cannot effectively deal with the new task identified in this paper. Actually, by proposing the new task of outlying subspace detection and HighDOD, we are able to have more options when handing outliers in high-dimensional space: when we want to detect outliers in a certain subspace, we can use the evolutionary-based search method, while when we are interesting in getting insights on the subspaces in which points are outliers, our technique can come into play.

Finally, we have to admit that the task of detecting outlying subspace for high-dimensional data itself still remains a very hard problem even after the search heuristics introduced in this paper have been applied. To further lower the hardness of this problem and make HighDOD more efficient, future research efforts will be taken in the following two directions. First, we will study the compress scheme for compacting the voluminous resultant outlying subspaces detected to make them more space efficient. Second, we are interested in exploring the parallel computing architecture to render HighDOD workable in a parallel paradigm so as to achieve a higher level of efficiency.

Acknowledgements The authors would like to thank Dr. Tok Wang Ling and Dr. Wynne Hsu, both from School of Computing, National University of Singapore, for their useful comments on the earlier drafts of this paper. This work is supported in part by NSERC Grant 312423.

References

1. Agrawal R, Gehrke J, Gunopulos D, Raghavan P (1998) Automatic subspace clustering of high dimensional data mining application. In: Proceedings of ACM SIGMOD'98, Seattle, Washington, USA, pp 94–105
2. Aggarwal CC, Procopiuc C, Wolf JL, Yu PS, Park JS (1999) Fast algorithms for projected clustering. In: Proceedings of the ACM SIGMOD'99, Philadelphia, Pennsylvania, USA, pp 61–72
3. Aggarwal CC, Yu PS (2001) Outlier detection in high dimensional data. In: Proceedings of the ACM SIGMOD'01, Santa Barbara, California, USA
4. Angiulli F, Pizzuti C (2002) Fast outlier detection in high dimensional spaces. In: Proceedings of PKDD'02, Helsinki, Finland, pp 15–26
5. Barnett V, Lewis T (1994) Outliers in statistical data, 3rd edn. Wiley, New York
6. Berchtold S, Keim DA, Kriegel H (1996) The X-tree: an index structure for high-dimensional data. In: Proceedings of the VLDB'96, Mumbai, India, pp 28–39
7. Breuning M, Kriegel H-P, Ng R, Sander J (2000) LOF: identifying density-based local outliers. In: Proceedings of the ACM SIGMOD'00, Dallas, Texas, pp 93–104

8. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the SIGKDD'96, Portland, Oregon, USA, pp 226–231
9. Han J, Kamber M (2000) Data mining: concepts and techniques. Morgan Kaufman
10. Hawkins D (1980) Identification of outliers. Chapman and Hall, London
11. Hinneburg A, Keim DA (1998) An efficient approach to cluster in large multimedia databases with noise. In: Proceedings of the SIGKDD'98, New York, NY, USA, pp 58–65
12. Jin W, Tung AKH, Han J (2001) Finding top *n* local outliers in large database. In: Proceedings of the SIGKDD'01, San Francisco, CA, pp 293–298
13. Knorr EM, Ng RT (1998) Algorithms for mining distance-based outliers in large dataset. In: Proceedings of the VLDB'98, New York, NY, pp 392–403
14. Knorr EM, Ng RT (1999) Finding intentional knowledge of distance-based outliers. In: Proceedings of the VLDB'99, Edinburgh, Scotland, pp 211–222
15. Mace AE (1964) Sample-size determination. Reinhold Publishing Corporation, New York
16. Ng R, Han J (1994) Efficient and effective clustering methods for spatial data mining. In: Proceedings of the VLDB'94, Santiago, Chile, pp 144–155
17. Papadimitriou S, Kitagawa H, Gibbons PB, Faloutsos C (2003) LOCI: fast outlier detection using the local correlation integral. In: Proceedings of the ICDE'03, Bangalore, India, pp 315–325
18. Preparata F, Shamos M (1998) Computational geometry: an introduction. Springer-Verlag, Berlin Heidelberg New York
19. Ramaswamy S, Rastogi R, Kyuseok S (2000) Efficient algorithms for mining outliers from large data sets. In: Proceedings of the ACM SIGMOD'00, Dallas, Texas, pp 427–438
20. Ruts I, Rousseeuw P (1996) Computing depth contours of bivariate point clouds. Comput Stat Data Anal 23: 153–168
21. Sarafis IA, Trinder PW, Zalzala AMS (2003) Towards effective subspace clustering with an evolutionary algorithm. In: IEEE congress on evolutionary computation, Canberra, Australia
22. Sheikholeslami G, Chatterjee S, Zhang A (1999) WaveCluster: a wavelet based clustering approach for spatial data in very large database. VLDB J 8(3/4): 289–304
23. Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the ACM SIGMOD'96, Montreal, Canada, pp 103–114

Ji Zhang received his BS from Department of Information Systems and Information Management at Southeast University, Nanjing, China, in 2000 and MSc from Department of Computer Science at National University of Singapore in 2002. He worked as a researcher in Center for Information Mining and Extraction (CHIME) at National University of Singapore from 2002 to 2003 and Department of Computer Science at University of Toronto from 2003 to 2005. He is currently with Faculty of Computer Science at Dalhousie University, Canada. His research interests include Knowledge Discovery and Data Mining, XML and Data Cleaning. He has published papers in *Journal of Intelligent Information Systems* (JIIS), *Journal of Database Management* (JDM), and major international conferences such as VLDB, WWW, DEXA, DaWaK, SDM, and so on.

Hai Wang is an assistant professor in the Department of Finance Management Science at Sobey School of Business of Saint Mary's University, Canada. He received his BSc in computer science from the University of New Brunswick, and his MSc and PhD in Computer Science from the University of Toronto. His research interests are in the areas of database management, data mining, e-commerce, and performance evaluation. His papers have been published in *International Journal of Mobile Communications*, *Data Knowledge Engineering*, *ACM SIGMETRICS Performance Evaluation Review*, *Knowledge and Information Systems*, *Performance Evaluation*, and others.